《第三章 程序设计基础》示例代码

作者：韩露露、杨波

日期：2019年3月1日

说明

本电子文档来源于书籍《深入浅出CryptoPP密码学库》，它最初被存放于GitHub上。任何人都可以复制、传播、使用本示例代码。

⇓

简介

《深入浅出CryptoPP密码学库》内容简介：

本书向读者介绍密码学库CryptoPP（或Crypto++）的使用方法和设计原理。CryptoPP是一个用C++语言编写的、开源的、免费的密码程序库，它最初由Wei Dai开发，现由开源社区维护。CryptoPP库广泛应用于学术界、开源项目、非商业项目以及商业项目，它几乎包括了目前已经公开的所有密码算法，支持当前主流的多种系统平台，并且具有良好的设计结构和较高的执行效率。

全书共15章，主要内容包括随机数发生器、Hash函数、流密码、分组密码、消息认证码、密钥派生和基于口令的密码、公钥加密系统、数字签名、密钥协商等，本书涵盖C++程序设计、设计模式、数论和密码学等知识。

本书最大的特点就是以应用为导向、以解决实际工程问题为目标，理论结合实践，将抽象的密码学变成保障信息安全的实际工具。

本书可以作为密码学、网络安全等专业在校学生的上机实验教材，也可以作为信息安全产品开发者、科研人员、密码算法实现者的参考手册。

⇓

资源

本书更多示例代码：https://github.com/locomotive-crypto
Crypto++网站：https://www.cryptopp.com/
Crypto++库GitHub地址：https://github.com/weidai11/cryptopp
Crypto++库SourceForge地址：https://sourceforge.net/projects/cryptopp/
Crypto++库Google论坛：
⇒公告通知地址：https://groups.google.com/forum/#!forum/cryptopp-announce
⇒用户群组地址：https://groups.google.com/forum/#!forum/cryptopp-users

目录

1 多态性（Polymorphism）和虚函数（Virtual Function） 1
1.1 虚函数引发多态性 1
1.2 虚析构函数的作用 3

2 使用SecByteBlock类 5

3 声明 7

1 多态性（Polymorphism）和虚函数（Virtual Function）

1.1 虚函数引发多态性

下面给出一个虚函数引发多态性的例子。

```
#include <iostream> //使用cin、cout
using namespace std; //使用C++标准命名空间std
class Cshape //基类
{
public:
  virtual void print() //在基类中定义虚函数print()
    {
        cout << "CShape" << endl;
    }
};
class CRectangle : public Cshape //CShape的派生类
{
public:
  virtual void print() //在派生类中重写虚函数print()
    {
        cout << "CRectangle." << endl;
    }
};
class CCricle :public Cshape //CShape的派生类
{
public:
  virtual void print() //在派生类中重写虚函数print()
    {
        cout << "CCricle." << endl;
    }
};
class CEllipse :public Cshape //CShape的派生类
{
public:
  virtual void print() //在派生类中重写虚函数print()
    {
        cout << "CEllipse." << endl;
    }
};
void Cprint(CShape* p) //以指向基类的指针作为函数的参数
{
    p->print(); //根据p的实际指向，执行对应的类的print函数
}
void CCprint(CShape& p) //以基类的引用作为函数参数
{
    p.print();
```

```
}
int main()
{
    //动态创建一个CCricle对象，并赋值给指向基类的指针
    CShape* pCircle = new CCricle();
    //动态创建一个CEllipse对象，并赋值给指向基类的指针
    CShape* pEllipse = new CEllipse();
    CRectangle Rectangle; //创建一个CRectangle对象
    Cprint(pCircle); //输出"CCricle."，pCircle实际指向的是CCricle对象
    Cprint(pEllipse); //输出"CEllipse."，pEllipse实际指向的是CEllipse对象
    //输出"CRectangle."，Rectangle实际是一个CRectangle对象
    CCprint(Rectangle);
    delete pCircle; //销毁动态创建的对象
    delete pEllipse; //销毁动态创建的对象
    return 0;
}
```

执行程序，程序的输出结果如下：

```
CCricle.
CEllipse.
CRectangle.
请按任意键继续. . .
```

1.2 虚析构函数的作用

在C++中，使用虚析构函数可以避免内存的泄漏。下面的示例程序演示了虚析构函数的这种作用。

```
#include<iostream> //使用cin、cout
using namespace std; //使用C++的标准命名空间std
class Cshape //基类
{
public:
  ~CShape() //非虚的析构函数
    {
        cout << "CShape析构函数被调用" << endl;
    }
};
class CCircle :public Cshape //派生类
{
public:
  ~CCircle()
    {
        cout << "CCircle的析构函数被调用" << endl;
    }
};
int main()
{
    CShape* pCShape = new CCircle(); //动态创建的CCircle对象
    delete pCShape; //通过指针删除动态创建的CCircle对象
    return 0;
}
```

执行程序，程序的输出结果如下：

```
CShape析构函数被调用
请按任意键继续. . .
```

从程序的执行结果可以发现，通过基类的指针去释放（delete）动态创建的派生类对象时，只有基类的析构函数被调用，而派生类的析构函数并没有被调用。

当把基类CShape的析构函数声明为虚函数时，

```
class Cshape //基类
{
public:
  virtual ~CShape() //非虚的析构函数
    {
        cout << "CShape析构函数被调用" << endl;
    }
};
```

再次执行程序，它的输出结果如下：

```
CCircle析构函数被调用
CShape析构函数被调用
请按任意键继续. . .
```

从程序输出结果可以看到，基类和派生类的析构函数都被调用。

2 使用SecByteBlock类

下面以SecBlock类模板最常用的实例–SecByteBlock类为例，演示它的使用方法：

```
#include<iostream> //使用cout、cin
#include<secblock.h> //使用SecByteBlock
using namespace std; //std是C++的命名空间
using namespace CryptoPP; //CryptoPP是CryptoPP库的命名空间
//作用：打印缓冲区str中的值
//参数str：待打印输出的缓冲区首地址
//参数length：缓冲区的长度
void Print(byte* str, size_t length);
int main()
{
    //定义一个SecByteBlock对象，并分配100个字节的存储空间
    SecByteBlock buf1(100);
    //将字符串"I like Cryptography"拷贝至buf1对象
    //SecByteBlock对象可以向void*自动转换：operator void *()
    memcpy(buf1, "I like Cryptography", strlen("I like Cryptography"
        ));
    //调用resize()函数重新分配内存大小
    //内存长度由100字节变为19字节
    buf1.resize(strlen("I like Cryptography"));
    cout << "buf1=";
    //SecByteBlock对象可以向byte*自动转换：operator T *()
    Print(buf1, buf1.size()); //打印buf1
    //调用拷贝构造函数，完成对象之间数据的拷贝
    //定义一个SecByteBlock对象，并将buf1对象的内容复制至buf2
    SecByteBlock buf2(buf1);
    cout << "buf2=";
    Print(buf2, buf2.size()); //打印buf2
    if (buf1 == buf2)
    {//重载运算符==：bool operator==(const SecBlock¡T, A¿ &t) const
        cout << "buf1 == buf2" << endl;
    }
    SecByteBlock buf3; //定义一个SecByteBlock对象
    //重载运算符=：SecBlock¡T, A¿& operator=(const SecBlock¡T, A¿ &t)
    buf3 = buf2;
    cout << "buf3=";
    Print(buf3, buf3.size()); //打印buf3
    SecByteBlock buf4; //定义一个SecByteBlock对象
    //重载运算符+：SecBlock¡T, A¿ operator+(const SecBlock¡T, A¿ &t)
    buf4 = buf1 + buf2 + buf3;
    cout << "buf4=";
    Print(buf4, buf4.size()); //打印buf4
    SecByteBlock buf5; //定义一个SecByteBlock对象
    //给buf5分配10个字节的存储空间，每个存储单元的值均为'V'
```

```
43      buf5.Assign(10, 'V');
44      cout << "buf5=";
45      Print(buf5, buf5.size()); //打印buf5
46      return 0;
47  }
48  void Print(byte* str, size_t length)
49  {//将缓冲区str指向的长度为length的数据以字符的形式输出
50      for (size_t i = 0; i < length; ++i)
51      {
52          cout << (char)str[i]; //强制类型转换后打印输出
53      }
54      cout << endl;
55  }
```

执行程序，程序的输出结果如下：

```
buf1=I like Cryptography
buf2=I like Cryptography
buf1 == buf2
buf3=I like Cryptography
buf4=I like CryptographyI like CryptographyI like Cryptography
buf5=VVVVVVVVVV
请按任意键继续. . .
```

3 声明

此为《深入浅出CryptoPP密码学库》随书电子文档，它仅包含书籍中示例程序的源代码。关于示例代码的解释说明，详见书籍相应章节内容。

由于作者水平有限，错误之处在所难免。欢迎通过如下方式反馈相关问题：

⇒ QQ：1220195669

⇒ 微信：cc1220195669